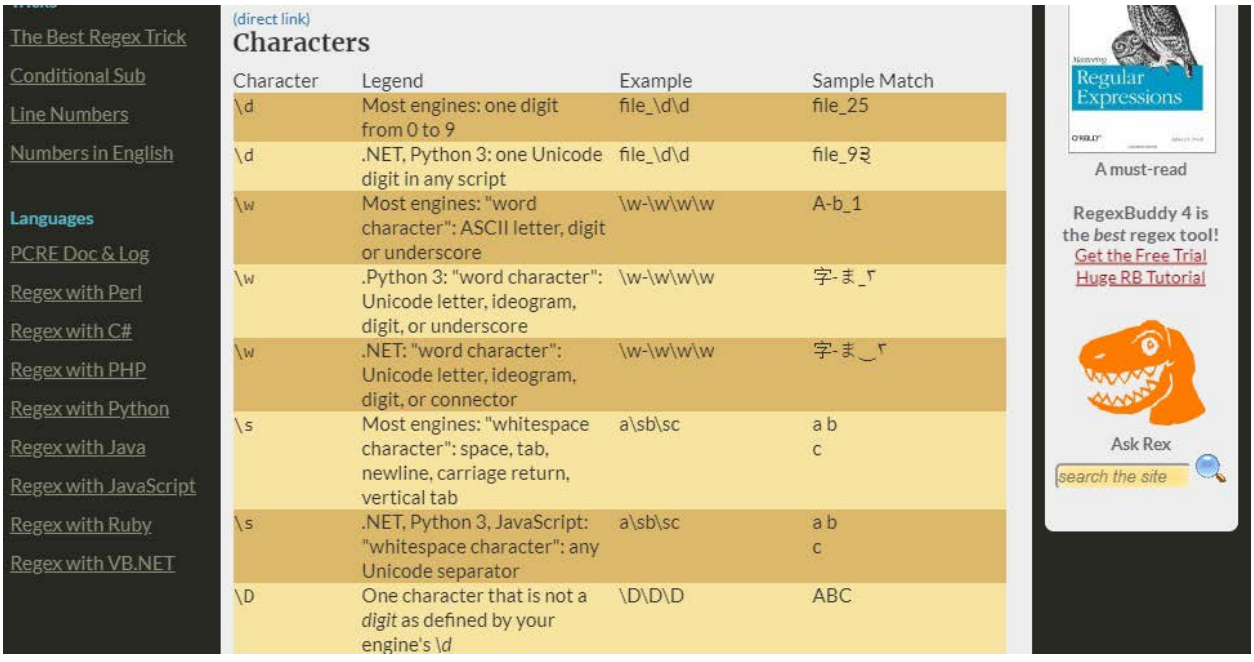
# Regular Expressions Guides

Here are a few links to RegEx guides favored by UNT catalogers (screenshots only cover a small portion of the resource):

## RexEgg

http://www.rexegg.com/regex-quickstart.html#ref

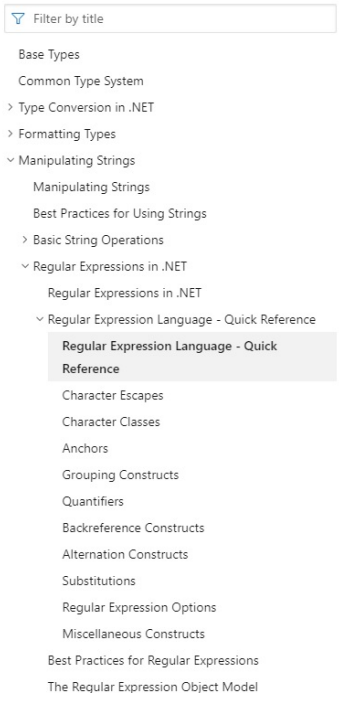



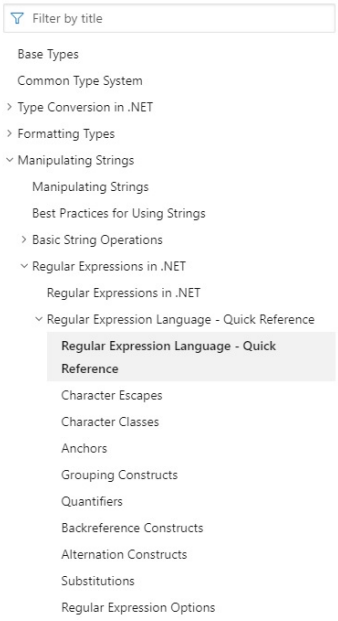


Jeremy Berg, UNT Libraries
Jeremy.berg@unt.edu

# Regular Expression Language Quick Reference

https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference




## Regular Expression Language - Quick Reference

03/29/2017 • 10 minutes to read • +11

A regular expression is a pattern that the regular expression engine attempts to match in input text. A pattern consists of one or more character literals, operators, or constructs. For a brief introduction, see .NET Regular Expressions.

Each section in this quick reference lists a particular category of characters, operators, and constructs that you can use to define regular expressions.

We've also provided this information in two formats that you can download and print for easy reference:

- Download in Word (.docx) format
- Download in PDF (.pdf) format

### Character Escapes

The backslash character (\) in a regular expression indicates that the character that follows it either is a special character (as shown in the following table), or should be interpreted literally. For more information, see Character Escapes.

| Escaped character | Description | Pattern | Matches |
|---|---|---|---|
| `\a` | Matches a bell character, \u0007. | `\a` | `"\u0007"` in `"Error!" + '\u0007'` |
| `\b` | In a character class, matches a backspace, \u0008. | `[\b]{3,}` | `"\b\b\b"` in `"\b\b\b"` |
| `\t` | Matches a tab, \u0009. | `(\w+)\t` | `"item1\t"`, `"item2\t"` in `"item1\titem2\t"` |

### Regular Expression Options

You can specify options that control how the regular expression engine interprets a regular expression pattern. Many of these options can be specified either inline (in the regular expression pattern) or as one or more RegexOptions constants. This quick reference lists only inline options. For more information about inline and RegexOptions options, see the article Regular Expression Options.

You can specify an inline option in two ways:

- By using the miscellaneous construct `(?imnsx-imnsx)`, where a minus sign (-) before an option or set of options turns those options off. For example, `(?i-mn)` turns case-insensitive matching (`i`) on, turns multiline mode (`m`) off, and turns unnamed group captures (`n`) off. The option applies to the regular expression pattern from the point at which the option is defined, and is effective either to the end of the pattern or to the point where another construct reverses the option.
- By using the grouping construct `(?imnsx-imnsx:` *subexpression* `)`, which defines options for the specified group only.

The .NET regular expression engine supports the following inline options:

| Option | Description | Pattern | Matches |
|---|---|---|---|
| `i` | Use case-insensitive matching. | `\b(?i)a(?-i)a\w+\b` | `"aardvark"`, `"aaaAuto"` in `"aardvark AAAuto aaaAuto Adam breakfast"` |
| `m` | Use multiline mode. `^` and `$` match the beginning and end of a line, instead of the beginning and end of a string. | For an example, see the "Multiline Mode" section in Regular Expression Options. | |
| `n` | Do not capture unnamed groups. | For an example, see the "Explicit Captures Only" section in Regular Expression Options. | |

# Working With Base Types In .Net

http://ccftp.scu.edu.cn:8090/Download/6bcfea8f-7b55-485c-8795-df92bed197bb.pdf
This PDF may also be downloaded from the above Quick Reference page.

## Contents



Console.WriteLine exposes the same functionality as String.Format. The only difference between the two methods is that String.Format returns its result as a string, while Console.WriteLine writes the result to the output stream associated with the Console object. The following example uses the Console.WriteLine method to format the value of `MyInt` to a currency value.

```
int MyInt = 100;
Console.WriteLine("{0:C}", MyInt);
// The example displays the following output
// if en-US is the current culture:
//       $100.00
```

```
Dim MyInt As Integer = 100
Console.WriteLine("{0:C}", MyInt)
' The example displays the following output
' if en-US is the current culture:
'       $100.00
```

The following example demonstrates formatting multiple objects, including formatting one object two different ways.

```
string myName = "Fred";
Console.WriteLine(String.Format("Name = {0}, hours = {1:hh}, minutes = {1:mm}",
      myName, DateTime.Now));
// Depending on the current time, the example displays output like the following:
//    Name = Fred, hours = 11, minutes = 30
```

```
Dim myName As String = "Fred"
Console.WriteLine(String.Format("Name = {0}, hours = {1:hh}, minutes = {1:mm}", _
                  myName, DateTime.Now))
```

# Dave Child's Regular Expressions Cheat Sheet

https://www.cheatography.com/davechild/cheat-sheets/regular-expressions/

## Regular Expressions Cheat Sheet by DaveChild

A quick reference guide for regular expressions (regex), including symbols, ranges, grouping, assertions and some sample patterns to get you started.

development regex regular expressions programming

### Anchors

| | |
|---|---|
| ^ | Start of string, or start of line in multi-line pattern |
| \A | Start of string |
| $ | End of string, or end of line in multi-line pattern |
| \Z | End of string |
| \b | Word boundary |
| \B | Not word boundary |
| \< | Start of word |
| \> | End of word |

### Character Classes

| | |
|---|---|
| \c | Control character |
| \s | White space |
| \S | Not white space |
| \d | Digit |
| \D | Not digit |

| | |
|---|---|
| [:lower:] | Lower case letters |
| [:alpha:] | All letters |
| [:alnum:] | Digits and letters |
| [:digit:] | Digits |
| [:xdigit:] | Hexadecimal digits |
| [:punct:] | Punctuation |
| [:blank:] | Space and tab |
| [:space:] | Blank characters |
| [:cntrl:] | Control characters |
| [:graph:] | Printed characters |
| [:print:] | Printed characters and spaces |
| [:word:] | Digits, letters and underscore |

### Quantifiers

| | | | |
|---|---|---|---|
| * | 0 or more | {3} | Exactly 3 |
| + | 1 or more | {3,} | 3 or more |
| ? | 0 or 1 | {3,5} | 3, 4 or 5 |

Add a ? to a quantifier to make it ungreedy.



### Escape Sequences

\ Escape following character

### Common Metacharacters

| | | | |
|---|---|---|---|
| ^ | [ | . | $ |
| { | * | ( | \ |
| + | ) | \| | ? |
| < | > | | |

The escape character is usually \

### Special Characters

| | |
|---|---|
| \n | New line |
| \r | Carriage return |
| \t | Tab |
| \v | Vertical tab |
| \f | Form feed |

### Groups and Ranges

| | |
|---|---|
| . | Any character except new line (\n) |
| (a\|b) | a or b |
| (...) | Group |
| (?:...) | Passive (non-capturing) group |
| [abc] | Range (a or b or c) |
| [^abc] | Not (a or b or c) |
| [a-q] | Lower case letter from a to q |
| [A-Q] | Upper case letter from A to Q |
| [0-7] | Digit from 0 to 7 |
| \x | Group/subpattern number "x" |

Ranges are inclusive.

### Pattern Modifiers

| | |
|---|---|
| g | Global match |
| i * | Case-insensitive |
| m * | Multiple lines |

| | |
|---|---|
| $n | nth non-passive group |
| $2 | "xyz" in /^(abc(xyz))$/ |
| $1 | "xyz" in /^(?:abc)(xyz)$/ |
| $` | Before matched string |
| $' | After matched string |
| $+ | Last matched string |
| $& | Entire matched string |

Some regex implementations use \ instead of $.